Finite elements are a good foundation for large-scale simulations on current and future architectures

- Backed by well-developed theory
- Naturally support unstructured and curvilinear grids.
- **Finite elements naturally connect different physics**

  ![Diagram showing connections between different physics](image)

  - High-order kinematics
  - High-order MHD
  - High-order rad. diffusion
  - High-order thermodynamics

- **High-order finite elements on high-order meshes**
  - increased accuracy for smooth problems
  - sub-element modeling for problems with shocks
  - bridge unstructured/structured grids
  - bridge sparse/dense linear algebra
  - HPC utilization, FLOPs/bytes increase with the order

- **Need new (interesting!) R&D for full benefits**
  - meshing, discretizations, solvers, AMR, UQ, visualization, …
Modular Finite Element Methods (MFEM)

**Flexible discretizations on unstructured grids**
- Triangular, quadrilateral, tetrahedral, hexahedral, prism; volume, surface and topologically periodic meshes
- Bilinear/linear forms for: Galerkin methods, DG, HDG, DPG, IGA, ...
- Local conforming and non-conforming AMR, mesh optimization
- Hybridization and static condensation

**High-order methods and scalability**
- Arbitrary-order H1, H(curl), H(div)- and L2 elements
- Arbitrary order curvilinear meshes
- MPI scalable to millions of cores + GPU accelerated
- Enables development from laptops to exascale machines.

**Solvers and preconditioners**
- Integrated with: HYPRE, SUNDIALS, PETSc, SLEPc, SUPERLU, VisIt, ...
- AMG solvers for full de Rham complex on CPU+GPU, geometric MG
- Time integrators: SUNDIALS, PETSc, built-in RK, SDIRK, ...

**Open-source software**
- Open-source (GitHub) with 114 contributors, 50 clones/day
- Part of FASTMath, ECP/CEED, xSDK, OpenHPC, E4S, ...
- 75+ example codes & miniapps: mfem.org/examples
Example 1 – Laplace equation

- **Mesh**
  - Read the mesh from the given mesh file. We can handle triangular, quadrilateral, tetrahedral, hexahedral, surface and volume meshes with the same code.
  - Mesh *mesh;
  - ifstream input(mesh_file);
  - if (input)
    - { cerr << "Can not open mesh file " << mesh_file << "\n" << endl;
      return 2;
    }
  - mesh = new Mesh(input, 1, 1);
  - input.close();
  - int dim = mesh->Dimension();
  - // 3. Refine the mesh to increase the resolution. In this example we do
  - // 'ref_levels' of uniform refinement. We choose 'ref_levels' to be the
  - // largest number that gives a final mesh with no more than 50,000
  - // elements.
  - for (int l = 0; l < ref_levels; l++)
    - mesh->UniformRefinement();

- **Finite element space**
  - Define a finite element space on the mesh. Here we use continuous
  - Lagrange finite elements of the specified order. If order < 1, we
  - instead use an isoparametric/isogeometric space.
  - FiniteElementCollection *fes;
  - if (order > 0)
    - fes = new H1_FECollection(order, dim);
  - else if (mesh->GetNElements() > 0)
    - fes = mesh->GetNElements();
  - else
    - fes = new H1_FECollection(order - 1, dim);
  - FiniteElementSpace *fespace = new FiniteElementSpace(mesh, fes);
  - const << "Number of unknowns: " << fespace->GetVSize() << endl;

- **Initial guess, linear/bilinear forms**
  - Set up the linear form b(\cdot) which corresponds to the right-hand side of
  - the FEM linear system, which in this case is \( l(\cdot, \phi; i) \) where \( \phi \) is
  - the basis functions in the finite element space.
  - LinearForm *b = new LinearForm(fespace);
  - ConstantCoefficient one(1.0);
  - b->AddDomainIntegrator(new DomainIntegrator(one));
  - b->Assemble();

- **Linear solve**
  - Define the solution vector u as a finite element grid function
  - corresponding to fespace. Initialize u with initial guess of zero,
  - which satisfies the boundary conditions.
  - GridFunction u(fespace);
  - x = 0.9;

- **Visualization**
  - works for any mesh & any H1 order
  - builds without external dependencies
Example 1 – Laplace equation

- **Mesh**

```cpp
63 // 2. Read the mesh from the given mesh file. We can handle triangular,
64 // quadrilateral, tetrahedral, hexahedral, surface and volume meshes with
65 // the same code.
66 Mesh *mesh;
67 ifstream imesh(mesh_file);
68 if (!imesh)
69 {
70   cerr << "Can not open mesh file: " << mesh_file << '\n' << endl;
71   return 2;
72 }
73 mesh = new Mesh(imesh, 1, 1);
74 imesh.close();
75 int dim = mesh->Dimension();
76
77 // 3. Refine the mesh to increase the resolution. In this example we do
78 // 'ref_levels' of uniform refinement. We choose 'ref_levels' to be the
79 // largest number that gives a final mesh with no more than 50,000
80 // elements.
81 {
82   int ref_levels =
83     (int)floor(log(50000./mesh->GetNE())/log(2.)/dim);
84   for (int l = 0; l < ref_levels; l++)
85     mesh->UniformRefinement();
86 }
```
Example 1 – Laplace equation

- Finite element space

```cpp
88 // 4. Define a finite element space on the mesh. Here we use continuous
89 //     Lagrange finite elements of the specified order. If order < 1, we
90 //     instead use an isoparametric/isogeometric space.
91 FiniteElementCollection *fec;
92 if (order > 0)
93     fec = new H1_FECollection(order, dim);
94 else if (mesh->GetNodes())
95     fec = mesh->GetNodes()->OwnFEC();
96 else
97     fec = new H1_FECollection(order = 1, dim);
98 FiniteElementSpace *fespace = new FiniteElementSpace(mesh, fec);
99 cout << "Number of unknowns: " << fespace->GetVSize() << endl;
```
Example 1 – Laplace equation

- Initial guess, linear/bilinear forms

```plaintext
101 // 5. Set up the linear form b(.) which corresponds to the right-hand side of
102 // the FEM linear system, which in this case is (1, phi_i) where phi_i are
103 // the basis functions in the finite element space fespace.
104 LinearForm *b = new LinearForm(fespace);
105 ConstantCoefficient one(1.0);
106 b->AddDomainIntegrator(new DomainLPIIntegrator(one));
107 b->Assemble();

108 // 6. Define the solution vector x as a finite element grid function
109 // corresponding to fespace. Initialize x with initial guess of zero,
110 // which satisfies the boundary conditions.
112 GridFunction x(fespace);
113 x = 0.0;

114 // 7. Set up the bilinear form a(.,.) on the finite element space
115 // corresponding to the Laplacian operator -Delta, by adding the Diffusion
116 // domain integrator and imposing homogeneous Dirichlet boundary
117 // conditions. The boundary conditions are implemented by marking all the
119 // boundary attributes from the mesh as essential (Dirichlet). After
120 // assembly and finalizing we extract the corresponding sparse matrix A.
121 BilinearForm *a = new BilinearForm(fespace);
122 a->AddDomainIntegrator(new DiffusionIntegrator(one));
123 a->Assemble();
124 Array<int> ess_bdr(mesh->bdr_attributes.Max());
125 ess_bdr = 1;
126 a->EliminateEssentialBC(ess_bdr, x, *b);
127 a->Finalize();
128 const SparseMatrix &A = a->SpMat();
```
Example 1 – Laplace equation

- Linear solve

```cpp
130  #ifdef MFEM_USE_SUITESPARSE
131     // 8. Define a simple symmetric Gauss-Seidel preconditioner and use it to
132     // solve the system Ax=b with PCG.
133     GSmoother M(A);
134     PCG(A, M, *b, x, 1, 200, 1e-12, 0.0);
135  #else
136     // 8. If MFEM was compiled with SuiteSparse, use UMFPACK to solve the system.
137     UMFPackSolver umf_solver;
138     umf_solver.Control[UMFPACK_ORDERING] = UMFPACK_ORDERING_METIS;
139     umf_solver.SetOperator(A);
140     umf_solver.Mult(*b, x);
141  #endif
```

- Visualization

```cpp
152     // 10. Send the solution by socket to a GLVis server.
153     if (visualization)
154     {
155         char vishost[] = "localhost";
156         int visport = 19916;
157         socketstream sol_sock(vishost, visport);
158         sol_sock.precision(8);
159         sol_sock << "solution\n" << *mesh << x << flush;
160     }
```
Example 1 – parallel Laplace equation

- **Parallel mesh**
  ```cpp
  // 5. Define a parallel mesh by a partitioning of the serial mesh. Define
  // this mesh further in parallel to increase the resolution. Once the
  // parallel mesh is defined, the serial mesh can be deleted.
  ParMesh *pmesh = new ParMesh(MPI_COMM_WORLD, *mesh);
  int par_ref_levels = 2;
  for (int l = 0; l < par_ref_levels; l++)
    pmesh->UniformRefinement();
  ```

- **Parallel finite element space**
  ```cpp
  ParFiniteElementSpace *fespace = new ParFiniteElementSpace(pmesh, fec);
  ```

- **Parallel initial guess, linear/bilinear forms**
  ```cpp
  ParLinearForm *b = new ParLinearForm(fespace);
  ParGridFunction x(fespace);
  ParBilinearForm *a = new ParBilinearForm(fespace);
  ```

- **Parallel linear solve with AMG**
  ```cpp
  // 11. Define and apply a parallel PCG solver for Ax=b with the BoomerAMG
  HypreSolver *solver = new HypreBoomerAMG(*A);
  HypreParCSR *vec = new HypreParCSR(*b);
  p0g->SetTol(1e-12);
  p0g->SetMaxIter(200);
  p0g->SetPrintLevel(2);
  p0g->SetPreconditioner(*solver);
  ```

- **Visualization**
  ```cpp
  // 14. Send the solution by socket to a GLVis server.
  ```

- **Parallel assembly**
  ```cpp
  A = P^T a P
  B = P^T b
  x = PX
  ```

- **highly scalable with minimal changes**
- **build depends on hypre and METIS**
Example 1 – parallel Laplace equation

```cpp
// 5. Define a parallel mesh by a partitioning of the serial mesh. Refine this mesh further in parallel to increase the resolution. Once the parallel mesh is defined, the serial mesh can be deleted.
ParMesh *pmesh = new ParMesh(MPI_COMM_WORLD, *mesh);
delete mesh;
{
    int par_ref_levels = 2;
    for (int l = 0; l < par_ref_levels; l++)
        pmesh->UniformRefinement();
}

ParFiniteElementSpace *fespace = new ParFiniteElementSpace(pmesh, fec);
ParLinearForm *b = new ParLinearForm(fespace);
ParGridFunction x(fespace);
ParBilinearForm *a = new ParBilinearForm(fespace);

// 10. Define the parallel (hypre) matrix and vectors representing a(.,..), b(.) and the finite element approximation.
HypreParMatrix *A = a->ParallelAssemble();
HypreParVector *B = b->ParallelAssemble();
HypreParVector *X = x.ParallelAverage();

// 11. Define and apply a parallel PCG solver for AX=B with the BoomerAMG preconditioner from hypre.
HypreSolver *amg = new HypreBoomerAMG(*A);
HyprePCG *pcg = new HyprePCG(*A);
```

```cpp
pcg->SetTol(1e-12);
```

```cpp
pcg->SetMaxIter(200);
```

```cpp
pcg->SetPrintLevel(2);
```

```cpp
pcg->SetPreconditioner(*amg);
```

```cpp
pcg->Mult(*B, *X);
```

```cpp
sol_sock << "parallel " << num_procs << " " << myid << "\n";
```

```cpp
sol_sock.precision(8);
```

```cpp
sol_sock << "solution\" << *pmesh << x << flush;
```
MFEM example codes: mfem.org/examples

- 40+ example codes, most with both serial + parallel versions
- Tutorials to learn MFEM features
- Starting point for new applications
- Show integration with many external packages
- Miniapps: more advanced, ready-to-use physics solvers
Some large-scale simulation codes powered by MFEM

- Inertial confinement fusion (BLAST)
- Topology optimization for additive manufacturing (LiDO)
- Core-edge tokamak EM wave propagation (SciDAC, RPI)
- MRI modeling (Harvard Medical)
- Heart modeling (Cardioid)
- Adaptive MHD island coalescence (SciDAC, LANL)
BLAST models shock hydrodynamics using high-order FEM in both Lagrangian and Remap phases of ALE

**Lagrangian phase (\( \vec{c} = \vec{0} \))**
- **Physical time evolution**
  - Based on physical motion

**Remap phase (\( \vec{c} = -\vec{v}_m \))**
- **Pseudo-time evolution**
  - Based on mesh motion

**Equation of Motion**
\[
\frac{d\vec{x}}{dt} = \vec{v}
\]

**Momentum Conservation**
\[
\frac{d}{dt} (\rho \vec{v}) = \vec{v} \cdot \vec{\sigma}
\]

**Mass Conservation**
\[
\frac{d\rho}{dt} = -\rho \nabla \cdot \vec{v}
\]

**Energy Conservation**
\[
\frac{d}{dt} (\rho e) = \vec{v} \cdot \nabla (\rho e)
\]

**Discont. Galerkin**
- Gauss-Lobatto basis

**Galerkin FEM**
- Bernstein basis
High-order finite elements lead to more accurate, robust and reliable hydrodynamic simulations

Parallel ALE for Q4 Rayleigh-Taylor instability (256 cores)

Robustness in Lagrangian shock-3pt axisymmm. interaction

Symmetry in 3D implosion

Symmetry in Sedov blast
High-order finite elements have excellent strong scalability

**Strong scaling, p-refinement**
- 1 zone/core
- ~600 dofs/zone
- Finite element partial assembly

**Strong scaling, fixed #dofs**
- 2D
- 256K DOFs
- 256 cores
- More FLOPs, same runtime

FLOPs increase faster than runtime
Conforming & Nonconforming Mesh Refinement

- **Conforming refinement**

  ![Conforming refinement](image)

- **Nonconforming refinement**

  ![Nonconforming refinement](image)

- **Natural for quadrilaterals and hexahedra**
Adaptive mesh refinement on library level:
- Conforming local refinement on simplex meshes
- **Non-conforming refinement for quad/hex meshes**
- \( h \)-refinement with fixed \( p \)

General approach:
- any high-order finite element space, \( H_1, H(\text{curl}), H(\text{div}), \ldots \), on any high-order curved mesh
- 2D and 3D
- arbitrary order hanging nodes
- anisotropic refinement
- derefinement
- serial and parallel, including parallel load balancing
- independent of the physics (easy to incorporate in applications)
General nonconforming constraints

$H(\text{curl})$ elements

Constraint: $e = f = d/2$

Indirect constraints

More complicated in 3D...

High-order elements

Constraint: local interpolation matrix

$s = Q \cdot m$, $Q \in \mathbb{R}^{9 \times 9}$
Nonconforming variational restriction

General constraint:

\[ y = Px, \quad P = \begin{bmatrix} I \\ W \end{bmatrix}. \]

- \( x \) – conforming space DOFs,
- \( y \) – nonconforming space DOFs (unconstrained + slave),
- \( \dim(x) \leq \dim(y) \)
- \( W \) – interpolation for slave DOFs

Constrained problem:

\[ P^TAPx = P^Tb, \]

\[ y = Px. \]
Nonconforming variational restriction
Nonconforming variational restriction

Regular assembly of A on the elements of the (cut) mesh
Nonconforming variational restriction

Conforming solution $y = P x$
AMR = smaller error for same number of unknowns

2D Shock-like Problem AMR Benchmark (Quad Mesh, Anisotropic Refinements)

- Uniform refinement
- 1st, 2nd, 4th, 8th order

Approximation error (H1 seminorm)

Square root of the number of unknowns

- order 1 uniform
- order 2 uniform
- order 4 uniform
- order 8 uniform
- order 1 aniso AMR
- order 2 aniso AMR
- order 4 aniso AMR
- order 8 aniso AMR

Anisotropic adaptation to shock-like fields in 2D & 3D
Parallel dynamic AMR, Lagrangian Sedov problem

Adaptive, viscosity-based refinement and derefinement. 2\textsuperscript{nd} order Lagrangian Sedov

Parallel load balancing based on space-filling curve partitioning, 16 cores
Parallel AMR scaling to ~400K MPI tasks

- weak+strong scaling up to ~400K MPI tasks on BG/Q
- **measure AMR only components**: interpolation matrix, assembly, marking, refinement & rebalancing (no linear solves, no “physics”)
The assembly/evaluation of FEM operators can be decomposed into **parallel**, **mesh topology**, **basis**, and **geometry/physics** components:

\[
A = P^T G^T B^T D B G P
\]

- **Partial assembly** = store only \( D \), evaluate \( B \) (tensor-product structure)
- Better representation than \( A \): *optimal memory*, near-optimal FLOPs
- Purely algebraic
- High-order operator format
- AD-friendly

*libCEED*, github.com/ceed/libceed
Example of a fast high-order operator

**Poisson problem** in variational form

Find \( u \in Q_p \subset H^1_0 \) s.t. \( \forall v \in Q_p \),

\[
\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v
\]

**Stiffness matrix** (unit coefficient)

\[
\int_\Omega \nabla \varphi_i \nabla \varphi_j = \sum_{E} \int_E \nabla \varphi_i \nabla \varphi_j
\]

\[
= \sum_{E} \sum_{k} \alpha_k J_E^{-1}(q_k) \nabla \hat{\varphi}_i(q_k) J_E^{-1}(q_k) \nabla \hat{\varphi}_j(q_k) \left| J_E(q_k) \right|
\]

\[
= \sum_{E} \sum_{k} \hat{\nabla} \hat{\varphi}_i(q_k) \left( \alpha_k J_E^{-T}(q_k) J_E^{-1}(q_k) \left| J_E(q_k) \right| \right) \hat{\nabla} \hat{\varphi}_j(q_k)
\]

\[
G, G^T, (B^T)_{ik}, D_{kk}, B_{kj}
\]

- \( J \) is the Jacobian of the element mapping (geometric factors)
- \( G \) is usually Boolean (except AMR)
- Element matrices \( A_E = B^T \mathbf{D} B \), are full, account for bulk of the physics, can be applied in parallel

\[
\begin{bmatrix}
A^1 \\
A^2 \\
\vdots \\
A^4
\end{bmatrix}
\]

- Never form \( A_E \), just apply its action based on actions of \( B, B^T \) and \( D \)
CEED BP1 bakeoff on BG/Q

All runs done on BG/Q (for repeatability), 16384 MPI ranks. Order $p = 1, \ldots, 16$; quad. points $q = p + 2$.

BP1 results of MFEM+xlc (left), MFEM+xlc+intrinsics (center), and deal.ii + gcc (right) on BG/Q.

Paper: “Scalability of High-Performance PDE Solvers”, IJHPCA, 2020

Cooperation/collaboration is what makes the bake-offs rewarding.
Device support in MFEM

MFEM support GPU acceleration in many linear algebra and finite element operations

- Several MFEM examples + miniapps have been ported with small changes
- Many kernels have a single source for CUDA, RAJA and OpenMP backends
- Backends are runtime selectable, can be mixed
- Recent improvements in CUDA, HIP, RAJA, SYCL, ...
MFEM performance on multiple GPUs

Single GPU performance: 2.6 GDOF/s
Problem size: 10+ million

Best total performance: 2.1 TDOF/s
Largest size: 34 billion

Optimized kernels for MPI buffer packing/unpacking on the GPU
Recent improvements on NVIDIA and AMD GPUs

New MFEM GPU kernels: perform on both V100 + MI100, have better strong scaling, can utilize tensor cores on A100, achieve 10+ GDOFs on H100

MI250X results in the CEED-MS39 report: ceed.exascaleproject.org/pubs
Matrix-free preconditioning

- **Explicit matrix assembly impractical at high order:**
  - Polynomial degree $p$, spatial dimension $d$
  - Matrix assembly + sparse matvecs:
    - $O(p^{2d})$ memory transfers
    - $O(p^{3d})$ computations
    - can be reduced to $O(p^{2d+1})$ computations by sum factorization
  - Matrix-free action of the operator (partial assembly):
    - $O(p^d)$ memory transfers – **optimal**
    - $O(p^{d+1})$ computations – **nearly-optimal**
    - efficient iterative solvers if combined with effective preconditioners

- **Challenges:**
  - Traditional matrix-based preconditioners (e.g. AMG) not available
  - Condition number of diffusion systems grows like $O(p^3/h^2)$
Low-Order-Refined (LOR) preconditioning

Efficient LOR-based preconditioning of $H^1$, $H(curl)$, $H(div)$ and $L^2$ high-order operators

- Pick LOR space and HO basis so $P=R=I$ (Gerritsma, Dohrmann)

- $A_{LOR}$ is sparse and spectrally equivalent to $A_{HO}$

**Theorem 2.** Let $M_\star$ and $K_\star$ denote the mass and stiffness matrices, respectively, where $\star$ represents one of the above-defined finite element spaces with basis as in Section 4.3. Then we have the following spectral equivalences, independent of mesh size $h$ and polynomial degree $p$.

$$
\begin{align*}
M_{V_h} &\sim M_{V_p}, & K_{V_h} &\sim K_{V_p}, \\
M_{W_h} &\sim M_{W_p}, & K_{W_h} &\sim K_{W_p}, \\
M_{X_h} &\sim M_{X_p}, & K_{X_h} &\sim K_{X_p}, \\
M_{Y_h} &\sim M_{Y_p-1}, & K_{Y_h} &\sim K_{Y_p-1}, \\
M_{Z_h} &\sim M_{Z_p}, & K_{Z_h} &\sim K_{Z_p}.
\end{align*}
$$

- $(A_{HO})^{-1} \approx (A_{LOR})^{-1} \approx B_{LOR}$ - can use BoomerAMG, AMS, ADS
High-order methods show promise for high-quality & performance simulations on exascale platforms

- **More information and publications**
  - MFEM – mfem.org
  - BLAST – computation.llnl.gov/projects/blast
  - CEED – ceed.exascaleproject.org

- **Open-source software**

- **Ongoing R&D**
  - GPU-oriented algorithms for Frontier, Aurora, El Capitan
  - Matrix-free scalable preconditioners
  - Automatic differentiation, design optimization
  - Deterministic transport, multi-physics coupling

Q4 Rayleigh-Taylor single-material ALE on 256 processors
Upcoming MFEM Events

MFEM in the Cloud Tutorial
August 10, 2023

MFEM Community Workshop
October 26, 2023

https://mfem.org/tutorial

https://mfem.org/workshop

Seminar series: https://mfem.org/seminar
This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
LLNL-PRES-755924

Disclaimer
This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.
Unstructured mesh – a spatial domain discretization composed of topological entities with general connectivity and shape

**Advantages**
- Automatic mesh generation for any level of geometric complexity
- Can provide the highest accuracy on a per degree of freedom basis
- General mesh anisotropy possible
- Meshes can easily be adaptively modified
- Given a complete geometry, with analysis attributes defined on that model, the entire simulation workflow can be automated

**Disadvantages**
- More complex data structures and increased program complexity, particularly in parallel
- Requires careful mesh quality control (level of control required is a function of the unstructured mesh analysis code)
- Poorly shaped elements increase condition number of global system – makes matrix solves harder
- Non-tensor product elements not as computationally efficient
Goal of FASTMath unstructured mesh developments include:

- Provide unstructured mesh components that are easily used by application code developers to extend their simulation capabilities
- Ensure those components execute on exascale computing systems and support performant exascale application codes
- Develop components to operate through multi-level APIs that increase interoperability and ease of integration
- Address technical gaps by developing tools that address needs and/or eliminate/minimize disadvantages of unstructured meshes
- Work with DOE application developers on integration of these components into their codes
Technology development areas:

- **Unstructured Mesh Analysis Codes** – Support application’s PDE solution needs – MFEM library is a key example
- **Performant Mesh Adaptation** – Parallel mesh adaptation to integrate into analysis codes to ensure solution accuracy
- **Dynamic Load Balancing and Task Management** – Technologies to ensure load balance and effectively execute by optimal task placement
- **Unstructured Mesh for Particle In Cell (PIC) Codes** – Tools to support PIC on unstructured meshes
- **Unstructured Mesh ML and UQ** – ML for data reduction, adaptive mesh UQ
- **In Situ Vis and Data Analytics** – Tools to gain insight as simulations execute
FASTMath Unstructured Mesh Tools and Components

### FE Analysis codes
- MFEM ([https://mfem.org/](https://mfem.org/))
- LGR ([https://github.com/SNLComputation/lgrtk](https://github.com/SNLComputation/lgrtk))
- PHASTA ([https://github.com/phasta/phasta](https://github.com/phasta/phasta))

### Unstructured Mesh Infrastructure
- Omega_h ([https://github.com/SNLComputation/omega_h](https://github.com/SNLComputation/omega_h))
- PUMI/MeshAdapt ([https://github.com/SCOREC/core](https://github.com/SCOREC/core))
- PUMIpic ([https://github.com/SCOREC/pumi-pic](https://github.com/SCOREC/pumi-pic))

### Load balancing, task placement
- Zoltan ([https://github.com/sandialabs/Zoltan](https://github.com/sandialabs/Zoltan))
- Xtra-PULP ([https://github.com/HPCGraphAnalysis/PuLP](https://github.com/HPCGraphAnalysis/PuLP))
- EnGPar ([http://scorec.github.io/EnGPar/](http://scorec.github.io/EnGPar/))

### Unstructured Mesh PIC applications
- XGCm ([https://github.com/SCOREC/xgcm](https://github.com/SCOREC/xgcm))
- GITRm ([https://github.com/SCOREC/gitrn](https://github.com/SCOREC/gitrn))
Parallel Unstructured Mesh Infrastructure

Support unstructured mesh interactions on exascale systems

• Mesh hierarchy to support interrogation and modification
• Maintains linkage to original geometry
• Conforming mesh adaptation
• Inter-process communication
• Supports field operations

Tools

• Omega_h – CPU/GPU support
• PUMI – CPU based curved mesh adapt.
• PUMIPic – Unstructured mesh with particles for CPU/GPU
Mesh Generation and Control

Mesh Generation:

- Automatically mesh complex domains – should work directly from CAD, image data, etc.
- Use tools like Gmsh, Simmetrix, etc.

Mesh control:

- Use *a posteriori* information to improve mesh
- Curved geometry and curved mesh entities
- Support full range of mesh modifications – vertex motion, mesh entity curving, cavity based refinement and coarsening, etc. anisotropic adaptation
- Control element shapes as needed by the various discretization methods for maintaining accuracy and efficiency

Parallel execution of all functions is critical on large meshes.
General Mesh Modification for Mesh Adaptation

- Driven by an anisotropic mesh size field that can be set by any combination of criteria
- Employ a “complete set” of mesh modification operations to alter the mesh into one that matches the given mesh size field
- Advantages
  - Supports general anisotropic meshes
  - Can obtain level of accuracy desired
  - Can deal with any level of geometric domain complexity
  - Solution transfer can be applied incrementally - provides more control to satisfy conservation constraints

**Diagram:**
- Edge split
- Face split
- Edge collapse
- Double split collapse to remove sliver
Mesh Adaptation Status

- Applied to very large scale models – 92B elements on 3.1M processes on ¾ million cores
- Local solution transfer supported through callback
- Effective storage of solution fields on meshes
- Supports adaptation with boundary layer meshes
Many applications have geometry that evolves as a function of the results – Effective adaptive loops combine mesh motion and mesh modification

Adaptive loop:
1. Initialize analysis case, generate initial mesh, start time stepping loop
2. Perform time steps employing mesh motion - monitor element quality and discretization errors
3. When element quality is not satisfactory or discretization errors too large – set mesh size field and perform mesh modification
4. Return to step 2.
Mesh Adaptation

- Supports adaptation of curved elements
- Adaptation based on multiple criteria, examples
  - Level sets at interfaces
  - Tracking particles
  - Discretization errors
  - Controlling element shape in evolving geometry
Load Balancing, Dynamic Load balancing

- **Purpose:** Balance or rebalance computational load while controlling communications
  - Equal “work load” with minimum inter-process communications
- **FASTMath load balancing tools**
  - Zoltan/Zoltan2 libraries provide multiple dynamic partitioners with general control of partition objects and weights
  - EnGPar diffusive multi-criteria partition improvement
  - XtraPuLP multi-constraint multi-objective label propagation-based graph partitioner
Reduce application communication time at extreme scale

Partitioning and load balancing: assign work to processes in ways that avoid process idle time and minimize communication

Task mapping: assign processes to cores/GPUs in ways that reduce messages distances and network congestion

Important in extreme-scale systems:
- Small load imbalances can waste many resources
- Large-scale networks can cause messages to travel long routes and induce congestion

Algorithms developed to:
- Account for underlying architectures & hierarchies
- Run effectively side-by-side with application across many platforms (multicore, GPU)
Zoltan/Zoltan2 Toolkits: Partitioners

Suite of partitioners supports a wide range of applications; no single partitioner is best for all applications.

**Geometric**
- Recursive Coordinate Bisection
- Recursive Inertial Bisection
- Multi-Jagged Multi-section

**Topology-based**
- PHG Graph Partitioning
- Interface to ParMETIS (U. Minnesota)
- Interface to PT-Scotch (U. Bordeaux)

**Space Filling Curves**

PHG Hypergraph Partitioning
- Interface to PaToH (Ohio St.)
EnGPar quickly reduces large imbalances on (hyper)graphs with billions of edges on up to 512K processes

- Multi-(hyper)graph supports multiple dependencies (edges) between application work/data items (vertices)
- Application defined vertex and edges
- Diffusion sending if work from heavily loaded parts to lightly loaded parts
- In 8 seconds, EnGPar reduced a 53% vtx imbalance to 6%, at a cost of 5% elm imbalance, and edge cut increase by 1% on a 1.3B element mesh
- Applied to PIC calculations to support particle balance – 20% reduction in total run time

Application of EnGPar particle dynamic load balancing in a GITRm impurity transport simulation
Creation of Parallel Adaptive Loops

Parallel data and services used to develop adaptive loop

- Geometric model topology for domain linkage
- Mesh topology – it must be distributed
- Simulation fields distributed over geometric model and mesh
- Partition control
- Dynamic load balancing required at multiple steps
- API’s to link to
  - CAD
  - Mesh generation and adaptation
  - Error estimation
  - etc.

### Diagram

- Parallel Data & Services
  - Domain Topology
  - Mesh Topology/Shape
  - Partition Control
  - Dynamic Load Balancing
- Solution Transfer
  - Mesh Generation and/or Adaptation
  - meshes and fields
  - mesh size field
  - Solution transfer constraints
  - PDE’s and discretization methods
- Correction Indicator
  - mesh with fields
- Mesh-Based Analysis
  - calculated fields
- Physics and Model Parameters
- Input Domain Definition with Attributes
  - non-manifold model construction
  - geometric interrogation
  - Attributed topology
  - geometry updates

### Additional Details

- API’s to link to:
  - CAD
  - Mesh generation and adaptation
  - Error estimation
  - etc.
In memory adaptive loops support effective data movement

In-memory adaptive loops for:

- MFEM – High order FE framework
- PHASTA – FE for NS
- FUN3D – FV CFD
- Proteus – multiphase FE
- Albany – FE framework
- ACE3P – High order FE electromagnetics
- M3D-C1 – FE based MHD
- Nektar++ – High order FE flow
Application interactions – Accelerator EM

Omega3P Electro Magnetic Solver (second-order curved meshes)

This figure shows the adaptation results for the CAV17 model. (top left) shows the initial mesh with ~126K elements, (top right) shows the final (after 3 adaptation levels) mesh with ~380K elements, (bottom left) shows the first eigenmode for the electric field on the initial mesh, and (bottom right) shows the first eigenmode of the electric field on the final (adapted) mesh.
Application interactions – Land Ice

- FELIX, a component of the Albany framework is the analysis code
- Omega_h parallel mesh adaptation is integrated with Albany to do:
  - Estimate error
  - Adapt the mesh
- Ice sheet mesh is modified to minimize degrees of freedom
- Field of interest is the ice sheet velocity
Application interactions – RF Fusion

- Accurate RF simulations require:
  - Detailed antenna CAD geometry
  - CAD geometry defeaturing
  - Extracted physics curves from GEQDSK equilibrium file
  - Analysis geometry combines CAD, and physics geometry
  - 3D meshes for accurate FE calculations in MFEM
  - Projection based error estimator
  - Conforming mesh adaptation with PUMI
Supporting Unstructured Mesh for Particle-in-Cell Calculations

PUMIPic data structures are mesh centric

- Mesh is distributed as needed by the application in terms of PICparts
- Mesh can be graded and anisotropic
- Particle data associated with elements
- Operations take advantage of distributed mesh topology
- Mesh distributed in PICparts
  - Start with a partition of mesh into a set of “core parts”
  - A PICpart is defined by a “core part” and sufficient buffer to keep particles on process for one or more pushes
  - GPU version defines buffer as set of neighboring parts
Mesh Data Structure for Heterogeneous Systems

- Mesh topology/adaptation tool - Omega
  - Conforming mesh adaptation (coarsening past initial mesh, refinement, swap)
  - Manycore and GPU parallelism using Kokkos
  - Distributed mesh via mesh partitions with MPI communications
  - Support for mesh-based fields

- Recent developments:
  - Curved mesh adaptation
  - More efficient field storage
  - Kokkos implementation on latest NVIDIA, AMD and Intel GPUs

Adaptation following rotating flow field.

Mesh entity adjacency arrays.

Serial and RIB partitioned mesh of RF antenna and vessel model.
PUMIPic Particle Data Structures

- Layout of particles in memory is critical to performance
  - Optimizes push (sort/rebuild), scatter, and gather operations
  - Associate particles with elements for large per element particle cases
  - Support changes in the number of particles per element
  - Evenly distributes work under a range of particle distributions (e.g. uniform, Gaussian, exponential, etc.)
  - Stores a lot of particles per GPU – low overhead

- Particle data structure interface and implementation
  - API abstracts implementation for PIC code developers
  - CSR, Sell-C-σ, CabanaM
  - Performance is a function of particle distribution
  - Cabana AoSoA w/a CSR index of elements-to-particles are promising

Left to Right: CSR, SCS with vertical slicing (yellow boxes), CabanaM (red boxes are SOAs). C is a team of threads.
PIC Operations Supported by PUMIPic

- Particle push
- Adjacency based search
  - Faster than grid based search
- Element-to-particle association update
- Particle Migration
- Particle path visualization
- Mesh partitioning w/buffer regions
- Mesh field association
- Poisson field solve using PETSc DMPIlex on GPUs
- Checkpoint/restart of particle and mesh data – supports customization for each application

XGCm is a version of XGC built on PUMIPic
- targeting execution of all operations on GPUs
Testing of PUMIPic for use in XGC like push
- 2M elements, 1M vertices, 2 to 128 poloidal planes
- Pseudo push and particle-to-mesh gyro scatter
- Tested on up to 24,576 GPUs of Summit with 1.1 trillion particles, for 100 iterations: push, adjacency
- PUMIPic weak scaling up to 24576 GPUs (4096 nodes) with 48 million particles per GPU

XGCm status: All operations on GPU
- Ion and electron scatter and push
- Electrostatic potential calculation
- Gyro-kinetic electric field calculation and gather
- Poisson solve

590,143 mesh elements, 20 million particles/GPU, \( n_{\text{planes}} = n_{\text{nodes}}/8 \)
PUMIPic based GITRm Impurity Transport Code

- Incorporates impurity transport capabilities of GITR
- 3D mesh for cases including divertors, tiles, limiters, specific diagnostics/probes etc.

Status
- Physics equivalent to GITR
- Particle initialization directly on 3D mesh
- 3D mesh design/control including anisotropy to properly represent the background fields
- Field transfer from SOLPS to 3D mesh
- Non-uniform particle distribution – evolves quickly in time
- Load balancing particles via EnGPPar
- Distance to boundary for sheath E field
- Post-processing on 3D unstructured mesh
Run the latest Simmetrix and PUMI software on RPI systems

We will help you run the latest Simmetrix and PUMI model preparation, mesh generation, and adaptation tools on your problem using HPC systems at RPI.

Contact Cameron Smith in Slack, during Speed-Dating, or via email at smithc11@rpi.edu for more information.